CSCE 210 LAB 2
EQUALITY AND THE COMPARABLE/ COMPARATOR INTERFACES

G. C. MUGANDA

1. Setting the stage

A Netbeans folder with two files is provided for you. One of the files contains a `Student` class with three fields as shown here.

```
public class Student
{
    String fname, lname;
    int [] testScores;
    public Student(String fname, String lname, int [] scores)
    {
        this.fname = fname;
        this.lname = lname;
        this.testScores = scores;
    }
    @Override
    public String toString()
    {
        String str = String.format("%s, %s scores: %s",
                                   lname, fname,
                                   Arrays.toString(testScores)
                                  );
        return str;
    }
}
```

The second file contains a main method that opens a file that has student data. You should create a text file with a "`.txt`" extension and store it in your home directory. On Windows machines, that will be your documents folder. Here is a sample input file:

```
5 3
Alan Quatermain 27  89  99
Betty Malandara  80 45 100
Sue Bolzano  34 89 67
Joseph Puff  34 67 21
Boris Johnson 90 34 89
```

The file starts off with an positive integer N, the number of student, and a positive integer T, the number of test scores. In this example, there are 5 students, and each student has 3 test scores.

Here is the second file. Its maim method opens the file, reads the student data into an array, and prints out the array.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import javax.swing.JFileChooser;
import javax.swing.filechooser.FileNameExtensionFilter;

public class CSCE210Lab2
{
    public static void main(String[] args)
                          throws FileNotFoundException
    {
        // Open file
        JFileChooser chooser = new JFileChooser();
        FileNameExtensionFilter filter =
           new FileNameExtensionFilter("Student data", "txt");
        chooser.setFileFilter(filter);
        int returnVal = chooser.showOpenDialog(null);
        if (returnVal != JFileChooser.APPROVE_OPTION)
        {
            System.out.println("No file specified.  Goodbye.");
            System.exit(1);
        }
        File inputFile = chooser.getSelectedFile();
        Scanner inputScanner = new Scanner(inputFile);

        // Read the file contents into an array of students
        Student [] students;
        // number of students
        int size =  inputScanner.nextInt();

        // number tests per student
        int numberOfTests = inputScanner.nextInt();
        students = new Student[size];

        // loop to read the data for all student
        for (int k = 0; k < size; k++)
        {
            // read the student data
            String fname, lname;
            int [] scores = new int[numberOfTests];
            fname = inputScanner.next();
            lname = inputScanner.next();
            for (int i = 0; i < numberOfTests; i++)
            {
                scores[i] = inputScanner.nextInt();
            }
            // create a new student object and add to array
            students[k] = new Student(fname, lname, scores);
        }
```

```
        // Output the array of student data
        System.out.println("Here is a list of all the students:");
        for(Student s : students)
        {
            System.out.println(s);
        }
    }
}
```

Running this program with the given file yields the following output

```
Here is a list of all the students:
Quatermain, Alan scores: [27, 89, 99]
Malandara, Betty scores: [80, 45, 100]
Bolzano, Sue scores: [34, 89, 67]
Puff, Joseph scores: [34, 67, 21]
Johnson, Boris scores: [90, 34, 89]
```

2. Add an `equals()` Method

Modify the `Student` class by overriding the `boolean equals(Object ob)` method to return true if and only if the passed parameter `ob` is considered equal to this object. Two `Student` objects are considered equal if they have the same first and last names.

The `equals()` method must work correctly for all of the following cases:

(1) the passed parameter is `null`.
(2) the passed parameter is not a `Student` object
(3) The passed parameter is a `Student` object

Add a loop to the main method that compares each element of the `students` array to

(1) `null`
(2) the string "`Sue Bolzano`"
(3) the object `new Student("Sue", "Bolzano", new int []{12})`

Do this by adding three sections at the end of main method, as indicated by the following comments:

```
// Test against null


// Test against the string object "Sue Bolzano"


// Test against a Student object
Student sueB = new Student("Sue", "Bolzano", new int []{12});
```

When you are done and you run the program, the output should look like this:

```
Here is a list of all the students:
Quatermain, Alan scores: [27, 89, 99]
Malandara, Betty scores: [80, 45, 100]
```

```
Bolzano, Sue scores: [34, 89, 67]
Puff, Joseph scores: [34, 67, 21]
Johnson, Boris scores: [90, 34, 89]
Testing equality against null:
false
false
false
false
false

Testing equality against the String 'Sue Bolzano':
false
false
false
false
false

Testing equality against the  object Bolzano, Sue scores: [12]
false
false
true
false
false
```

3. Implementing the `Comparable` interface

Modify the `Student` to implement the `Comparable<Student>` interface to order students by last name, and then by first name if two students have the same last name.

Create a second input file `students2.txt` by copying the first file and then adding the following line at the end of the file. Make sure you change the first line of the file from 5 3 to 6 3. The contents of the new file should look like this

```
6 3
Alan Quatermain 27  89  99
Betty Malandara  80 45 100
Sue Bolzano  34 89 67
Joseph Puff  34 67 21
Boris Johnson 90 34 89
Amy Bolzano  100 78 56
```

Now modify the main method by adding code at the end to sort the array of students, and then print sorted array. Use the `Arrays.sort()` method.

Your output should now look like this:

```
Here is a list of all the students:
Quatermain, Alan scores: [27, 89, 99]
Malandara, Betty scores: [80, 45, 100]
Bolzano, Sue scores: [34, 89, 67]
Puff, Joseph scores: [34, 67, 21]
Johnson, Boris scores: [90, 34, 89]
Bolzano, Amy scores: [100, 78, 56]
```

```
Testing equality against null:
false
false
false
false
false
false

Testing equality against the String 'Sue Bolzano':
false
false
false
false
false
false

Testing equality against the  object Bolzano, Sue scores: [12]
false
false
true
false
false
false

Here is the list of students in alphabetic order:
Bolzano, Amy scores: [100, 78, 56]
Bolzano, Sue scores: [34, 89, 67]
Johnson, Boris scores: [90, 34, 89]
Malandara, Betty scores: [80, 45, 100]
Puff, Joseph scores: [34, 67, 21]
Quatermain, Alan scores: [27, 89, 99]
```

4. Implementing the `Comparator` Interface

A comparator for the Java class `Student` is an object that compares two `Student` objects x_1 and x_2 and returns an integer that is negative, or 0, or positive to indicate whether x_1 comes before x_2, or x_1 is equal to x_2 or x_1 comes after x_2.

Whereas `Comparable<T>` builds a natural order into objects of the class `T`, the `Comparator<T>` interface compares two `T` objects from the outside. A class can only have one natural order, but you can have as many comparators for the class as you like. Comparators allow multiple ways of comparing objects.

The simplest way to make a comparator is to use a *lambda expression*. You can think of a lambda expression as an object of a class that defines only one method.

Suppose that we want to order the array of students in non-increasing order of their scores on the first test. In comparing two students $s1$ and $s2$ we want to return a negative integer if, on the first test, the score of $s1$ is larger than the score of s_2, 0 if the scores are equal, and a positive integer if the score of s_1 is less than that of s_2. Our lambda expression becomes

```
(s1, s2) -> { return s2.testScores[0] - s1.testScores[1]; }
```

To use a comparator to sort, you call `Arrays.sort(arr, c)` where `arr` is the array you want to sort and `c` is the comparator that specifies the sort order.

We can create an object of type `Comparator<Student>`, and then use it to sort our array of student like this

```
 Comparator<Student> score1Comparator =
                  (s1, s2) ->
                  {
                      return s2.testScores[0] - s1.testScores[0];
                  };
Arrays.sort(students, score1Comparator);
```

Once we have done this, we can print the sorted array. If we add this code to the end of our `main` method and then print, we get the following

```
Here is a list of all the students:
Quatermain, Alan scores: [27, 89, 99]
Malandara, Betty scores: [80, 45, 100]
Bolzano, Sue scores: [34, 89, 67]
Puff, Joseph scores: [34, 67, 21]
Johnson, Boris scores: [90, 34, 89]
Bolzano, Amy scores: [100, 78, 56]
Testing equality against null:
false
false
false
false
false
false

Testing equality against the String 'Sue Bolzano':
false
false
false
false
false
false

Testing equality against the  object Bolzano, Sue scores: [12]
false
false
true
false
false
false

Here is the list of students in alphabetic order:
Bolzano, Amy scores: [100, 78, 56]
Bolzano, Sue scores: [34, 89, 67]
Johnson, Boris scores: [90, 34, 89]
Malandara, Betty scores: [80, 45, 100]
Puff, Joseph scores: [34, 67, 21]
Quatermain, Alan scores: [27, 89, 99]
```

```
Here is the list of students  by score on first test:
Bolzano, Amy scores: [100, 78, 56]
Johnson, Boris scores: [90, 34, 89]
Malandara, Betty scores: [80, 45, 100]
Bolzano, Sue scores: [34, 89, 67]
Puff, Joseph scores: [34, 67, 21]
Quatermain, Alan scores: [27, 89, 99]
```

5. Writing your own comparator

Now delete the code you just added to sort by the first test score, and replace it by code that sorts by the sum of the scores. Modify your program so that after the array is sorted, you print each element in order of the total sum of scores, printing for each student the sum of that student's scores. The best way to do this is to add to the `Student` class a `getSumOfScores()` method.

Your output should look like this:

```
 Here is a list of all the students:
Quatermain, Alan scores: [27, 89, 99]
Malandara, Betty scores: [80, 45, 100]
Bolzano, Sue scores: [34, 89, 67]
Puff, Joseph scores: [34, 67, 21]
Johnson, Boris scores: [90, 34, 89]
Bolzano, Amy scores: [100, 78, 56]
Testing equality against null:
false
false
false
false
false
false

Testing equality against the String 'Sue Bolzano':
false
false
false
false
false
false

Testing equality against the  object Bolzano, Sue scores: [12]
false
false
true
false
false
false

Here is the list of students in alphabetic order:
Bolzano, Amy scores: [100, 78, 56]
Bolzano, Sue scores: [34, 89, 67]
```

```
Johnson, Boris scores: [90, 34, 89]
Malandara, Betty scores: [80, 45, 100]
Puff, Joseph scores: [34, 67, 21]
Quatermain, Alan scores: [27, 89, 99]

Here is the list of students  by score on first test:
Bolzano, Amy scores: [100, 78, 56]; Sum is 234
Malandara, Betty scores: [80, 45, 100]; Sum is 225
Quatermain, Alan scores: [27, 89, 99]; Sum is 215
Johnson, Boris scores: [90, 34, 89]; Sum is 213
Bolzano, Sue scores: [34, 89, 67]; Sum is 190
Puff, Joseph scores: [34, 67, 21]; Sum is 122
```

6. Submission

This lab is due Friday at the end of week 4.

You must submit a full Netbeans folder as required. Submissions that do not comply with this requirement will not be graded.